REMARKS

This is in response to the Office Action mailed February 14, 2003.

Applicant thanks the Examiner for the indication that Claims 16-17 are allowable.

The Examiner maintained the rejection of Claims 1-10 and 13-15 as being anticipated by or unpatentable over Alcorn et al. (USPN 5,643,086). Applicant thanks the Examiner for the detailed explanation regarding the rejection of these claims in light of Applicant's argument that Alcorn does not teach "encrypting" program code.

The Examiner asserts that Alcorn discloses performing a hash function upon game program code, which hash function is tantamount to an "encryption" of the program code. As Applicant understands the Examiner's position, it is that a "hash function" is a subset or form of "encryption." As detailed in the specification, Applicant's invention is a simplified method for protecting operation/control code in which the code is encrypted in a symmetrical encryption process. As detailed in the application, Applicant's method is accomplished using a form of encryption which allows the code to be recovered for use upon its decryption. In this respect, Applicant's invention differs substantially from that which occurs when a hash function is utilized.

As is known, when a hash function is performed upon data, the result of the hash function can not be used to recover the original data. As indicated at Math World (http://mathworld.wolfram.com), **hash functions are not reversible** (emphasis added, see Exhibit A hereto). As such, if a hash function is performed upon operating code, the output of the hash is irreversible, and while the output represents the operating code, the operating code can not be retrieved from the output of the hash function.

Independent Claims 1, 5 and 10 are directed to methods and devices in which control or operating code is encrypted and is then decrypted and used to control a gaming device. As claimed, the encryption must not be a hash function, since the output of the hash function could not be decrypted to re-obtain the operating or control code for use.

To further define the claims over the prior art, however, Applicant has amended independent Claims 1, 5 and 10. In particular, Applicant has amended the claims to recite that the encryption is "reversible" (again, clearly distinguishing from "hashing," which is not reversible), and reciting that the operating or control code is recovered for use (which is not possible if the code has been hashed).

Summary

Applicant asserts that Claims 1-10 and 13-17 are in a condition for allowance and respectfully requests a notice as to the same. If any matters remain outstanding, the Examiner is invited to contact the undersigned by telephone.

Respectfully submitted,

Dated: October 8, 2003

By: _____

R. Scott Weide
Registration No. 37,755
Weide & Miller, Ltd.
7251 W. Lake Mead Blvd., Suite 530
Las Vegas, NV 89128
(702)-382-4804 (Pacific time)

rsw-13265

EXHIBIT A

A hash function H projects a value from a set with many (or even an infinite number of) members to a value from a set with a fixed number of (fewer) members. Hash functions are not reversible. A hash function H might, for instance, be defined as $y = H(x) = \lfloor 10x \pmod 1 \rfloor$, where $x \in \mathbb{R}$, $y \in [0, 9]$, and $\lfloor x \rfloor$ is the floor function.

Hash functions can be used to determine if two objects are equal (possibly with a fixed average number of mistakes). Other common uses of hash functions are checksums over a large amount of data (e.g., the cyclic redundancy check [CRC]) and finding an entry in a database by a key value. The UNIX c-shell (csh) uses a hash table to store the location of executable programs. As a result adding new executables in a user's search path requires regeneration of the hash table using the `rehash` command before these programs can be executed without specifying the complete path.

To illustrate the use of hash functions in database lookups, consider a database consisting of an array containing an index n, a name, and a telephone number, with names listed in arbitrary order.

n	Name	Number
0	Parker	12345
1	(empty)	
2	Davis	43534
3	Harris	32452
4	Corea	46532
5	Hancock	96562
6	Brecker	37811
7	(empty)	
$N-1$	Marsalis	54323

To look up Hancock from this array, you would start at the beginning of the array, compare the names, then try the next until the names match. This very simple algorithm finds any entry in 1 to N steps, giving an average seek time of $N/2$. The seek time is therefore proportional to N. A much faster result can generally be achieved, if the database is sorted.

n	Name
0	Brecker
1	Corea
2	Davis
3	Hancock
4	Harris
5	Marsalis